Division 3 Clinic

In the little bag you got:

1x	ATmega328P Nano V3 Controller Board Compatible Arduino
1x	USB 2.0 A Male to Mini-B
1x	Servo motor
1x	Breadboard
10x	1k ohm resistors to limit their current
	(LEDs: Vf < 5V, mcd > 25, tested at 20mA)
2x	3mm Red LED
2x	3mm Yellow LED
2x	3mm Green LED
2x	3mm Blue LED
2x	3mm White LED
1x	9V Battery connector wire
1x	*Light sensor (arrived, ask for it)*
1x	*Push button (arrived, ask for it)*
2x	N channel FETs
3x	Wire, female-female
3x	Wire, male-female
3x	Wire, male-male
1x	Bag, Ziploc

First thing you want to do is plug the Nano into your computer using the supplied USB cable. We found a few bad cables and need to fix them if you do not have another Mini USB cable sitting around. Once plugged into your PC, a red LED should be **on** and the one next to it should be **flashing**. The other two LEDs are only busy when you upload code, remember to watch them when uploading.

Secondly, try to install the **Arduino software** and **possibly the Windows device driver** or (**Mac device driver for older Macs, follow this link**) as mentioned at the top of the previous page. (you might only needed it, if you have a Windoze computer. If you **right-click** on **My Computer**, and press '**g**' for **Manage**, you will see "**Device Manager**" before Storage and if you see any **Yellow icon'd "USB 2.0 xxx"** device, you would need to **install** the **driver**. Download the file, unzip to some **location** and in the **Device Manager right click** on the Yellow device and select "**Update Driver**" - from your own computer (don't search the web) and selected the **location** where you unzipped it. In Windows 8 it takes almost forever and Windows 7 does it in a hurry, so get a cup of of coffee, or just a water depending on your operating session. For older Macs, and maybe newer Macs, please see the instructions in the link above.

When this is done, or the driver was already handy, you will now have a COM**4** or other number, and in the **Arduino IDE**, you need to select this port under the **Tools->Port** menu. The other setting to change (once) is the **Tools->Board** to "**Arduino Nano**" ... now you can open **File->Examples->01.Basics->Blink** and make some changes to the number in delay(**1000**); at the bottom. Now you can simply click the "**upload**" (2nd) icon at the top (or press **Ctrl-U**) and see the magic change on the board. First compile takes long and then a lot shorter on the next go. The blink program turns the LED on for 1 second and off for 1 second, which might be different than the blink already loaded, we also tested some Nano's so your LED might be doing something way different too!

What does the blink program do?

In the setup() function, it tells the ATmega328 processor to make pin 13 an output pin, this is where the red LED (and its resistor) is connected to. The setup() function only runs one time after power up. So we only do "setup" things here.

In the loop() function, we put code that is going to loop and loop forever! So, in blink.ino, they do a digitalWrite(13, HIGH) to output 5V to pin 13, which turns the LED on. Then they delay the CPU for 1000 milliseconds (while nothing else in your code is running, since we have no interrupt routines set up), then they make pin 13 go to 0V or LOW and the LED turns off. And again, we delay another second before the loop exits just for the underlying framework to start it again. So, back to the LED turning on. Follow?

What would this do?

```
void loop() {
  digitalWrite( 13, HIGH );   // turn the LED on (HIGH is the voltage level,
5.0V on our Nano (you also get 3.3V Arduinos))
  delay( 100 );               // wait for ...
  digitalWrite( 13, LOW );    // turn the LED off by making the voltage LOW
  delay( 100 );               // wait for ...
  digitalWrite( 13, HIGH );   // turn the LED on (HIGH is the voltage level)
  delay( 100 );               // wait for ...
  digitalWrite( 13, LOW);     // turn the LED off by making the voltage LOW
  delay( 700 );               // wait for 0.7 seconds
} // loop
```

Yes, a double flash, all done in 1 second!

or this?

```
void loop() {
  digitalWrite( 13, HIGH ); // turn the LED on (HIGH is the voltage level)
  delay( random( 100 ) );   // wait for a ..... second
  digitalWrite( 13, LOW );  // turn the LED off by making the voltage LOW
  delay( random( 100 ) );   // wait for a ..... second
} // loop
```

When you save the file as something else (avoid spaces), it will be shown in the future under File->Sketchbook->YourFileName instead of Examples.

Ok, so that is one LED, how 'bout some more...let's get to adding a few more...

When you do a search for "arduino nano v3 pinout" you will see diagrams with more numbers on them than we need to know, some are the actual pin number of the Atmel ATmega328chip itself, then there is the D1, D2 and A1, A2 ramblings, and some will give you the number of the pin as you count them around the edge of the board. So, a quick look at the **Nano Schematic** will show that we indeed need to deal with the ramblings! We already controlled pin 13, and that is labelled as D13/SCK, since it has the LED on! It is pin number 17 on the ATmega328 and pin number 16 counting around the board edge...yes, this WILL get confusing, or maybe counting is not my thing!

Pin D0 and D1 are connected to the USB to Serial chip (FTDI FT232RL or the other CH340 chip), so avoid using these pins, until you "upgrade" to the cheaper Arduino Pro Mini that does not have the USB chip on-board. At a later point, we will cover how to **program those using a Nano**.

So, D2, D3 and D4 looks like prime candidates for our traffic light (Green, Yellow and Red):

// this goes into setup()

```
  pinMode( 2, OUTPUT );
  pinMode( 3, OUTPUT );
  pinMode( 4, OUTPUT );
```

but, if we want to make coding a little easier down the line, we rather do before setup():

```
#define GREEN  2
#define YELLOW 3
#define REDn   4
```

and in setup()

```
  pinMode( GREEN,  OUTPUT );
  pinMode( YELLOW, OUTPUT );
  pinMode( RED,    OUTPUT );
```

this will make it easier to do the digitalWrites with, instead of knowing that it was pin 4:

```
void loop() {
  digitalWrite( RED, HIGH );   // turn the LED on (HIGH is the voltage level,
5.0V on our Nano (you also get 3.3V Arduinos))
  delay( 1000 );               // wait for 1s
  digitalWrite( RED, LOW );    // turn the LED off by making the voltage LOW
  delay( 1000 );               // wait for 1s
} // loop
```

Noticed that pin 13, or the on-board LED is not doing anything, so you might write some code for him too, just so you have a heart beat on the board giving you some indication that the code is running.

Ok, now you need to connect the REAL LEDs to pin 2, 3 and 4, marked as D2, D3 and D4. We are using a HIGH to turn each LED on, so the other end of the LED-resistor pair needs to be GND. (You might want to change this in the future, since most micro-controllers can **sink** more current than they can **source**, by driving the pin HIGH to turn the LED on, we are **sourcing** the current. So either the resistor first, or the LED first, from pin D2 to LED's long pin, then the short pin to the 1k ohm resistor and the other end of the resistor to a common GND row on the bread board, which we need to connect back to the Nano as well. Same for the LEDs from pin D3 and D4. Each get their own resistor, which you might want to adjust in value, if you feel they are not dim or right enough, compared to the other two. Also keep in mind that only one LED will be on here at any time, so it would be hard to compare. There is also a theory that the human eye and ear can not distinguish the difference between the power of two light sources or sound sources if they are not more than 2dB in power different and not on at the same time. That is why I stick to 1k Ohm!

And now for the LED questionnaire answer, a Red, Blue, Green and Yellow LED are made from different chemical substances, and according to **Wikipedia** here: "*When a suitable **voltage** is applied to the leads (of an LED), **electrons are able to recombine with electron holes** within the device, **releasing energy** in the form of **photons**. This effect is called **electroluminescence**, and the color of the light (corresponding to the **energy of the photon**) is determined by the **energy band gap of the semiconductor**.*" - and these energy band gaps are different in the different materials where a PN junction is formed as a diode. So, first, since it is a diode, it will only allow current to flow in one direction. A diode has an **Anode** and a **Cathode**, and the current (not the electrons) will only flow through a diode from the anode to the cathode. **Alphabetical order**, A to C, easy to remember now. Some manufacturers like to stick to some rule to make the anode pin longer and some don't, sorry. But each device will have a datasheet and if they want their product to be used, they will confess where the anode and cathode is in that document.

Red LEDs are made from Aluminium gallium arsenide (AlGaAs), Gallium arsenide phosphide (GaAsP), Aluminium gallium indium phosphide (AlGaInP) or Gallium(III) phosphide (GaP) and Blue LEDs are made from Zinc selenide (ZnSe), Indium gallium nitride (InGaN) or Silicon carbide (SiC) as substrate. The blue took chemical engineers a lot longer to discover, that is why blue LEDs were not that common 15 years ago and still a little more expensive than Red ones. White LEDs require a broad spectrum of energy to be transmitted, so you can make a White light by combining a Red, Green, and Blue (RGB, sound familiar) LEDs, but that is bulky and have Vf differences as described later, so most White LEDs look Yellow when off because that is the color of the phosphor that emits the wide spectrum "white" light when it is excited by blue or UV light, so again, you can see why white LEDs were not around 15 years ago either!

There are two more things to remember though, if you exceed the Vr or reverse voltage of a diode it will break down, in other words the diode would block the flow of current in the reverse direction, until Vr is exceeded, and then my friends, you might NOT have a diode or LED any more, so beware. The second thing to know, is a diode can not limit the amount of current flowing through it, and at some point, it will overheat and do a short white flash before heading to heaven. And this is why you need a current limiting resistor! So here is the math while we are at it. An LED will start to allow current to flow as soon as you exceed its Vf (V-forward) voltage. This voltage is different for different materials, so a blue LED would typically need more voltage than a red one to turn on.

The red, 627nm, **Kingbright LED in your kit** is rated with a typical Vf=2V, a Vr=5V and a max DC forward current of 30mA (160mA peak, but don't do that one for long!)

From the first graph on page 3, at 5mA, the forward voltage drop would be 1.8V...so, if we have a 5V source, we subtract the 1.8V from it, since that will sit across the diode, and we calculate the size of the resistor to conduct the 5mA. Since **V = I x R**, so 5-1.8 = 0.005 x R, and R = 3.2/0.005 = **640 Ohms**. I like my LEDs to last a long time, so anything from 2mA (1.65k Ohms for 1.7V@2mA) to 4mA (806 Ohms for 1.775V@4mA) is going to make my 50 year requirement, so a 1k Ohm resistor is what I use! Almost always. Some blue LEDs have Vf=3.2V so a 450 Ohm might give you 4mA and again I use a 1k Ohm resistor, but two of them in parallel, so I have 500 Ohms!

Does the "current limiting resistor" make a little more sense now? You don't want to use a single resistor with two or more LEDs, especially when they are not from the same manufacturer, chemistry, date code or bin. Since the Vf will be different, one of the LEDs will allow more current to flow through it, and it will heat up and die first. When that LED goes, now the single resistor still allows the same current to flow and the other LEDs are carrying the dead LEDs current too, which mean the next lowest Vf LED will die a whole lot sooner!!! You also run the risk of one LED clamping the voltage across them to a voltage too low for the others to turn on.

As a sharing resistor example, to have 5mA go through 3 Red LEDs, you might say I'll use a single 220 Ohm resistor to allow only 15mA to flow. 3 x 0.005 x 220 = 3.3V and 5-3.3 = 1.7V, most likely enough to have all three on. If one LED dies, for whatever reason, physically, lightning, or most likely heat, now you have two LEDs with 1.7V across them and a resistor with 3.3V across it, carrying 15mA (3.3/220=0.15) and guess what, each of the two LED is now carrying 7.5mA and not 5mA any more. 1.5 times more than planned! A resistor is 10 cents, two more will not hurt you, if you keep in mind that fixing this problem in 5 years, will be at least 1/2 hour's work, and what is that rate? Compared to the 20 cents you saved?

Did you know you can write and compile **code for your Arduino online**?

Did you know there are also **simulators for Arduino** online**like this one**and others?